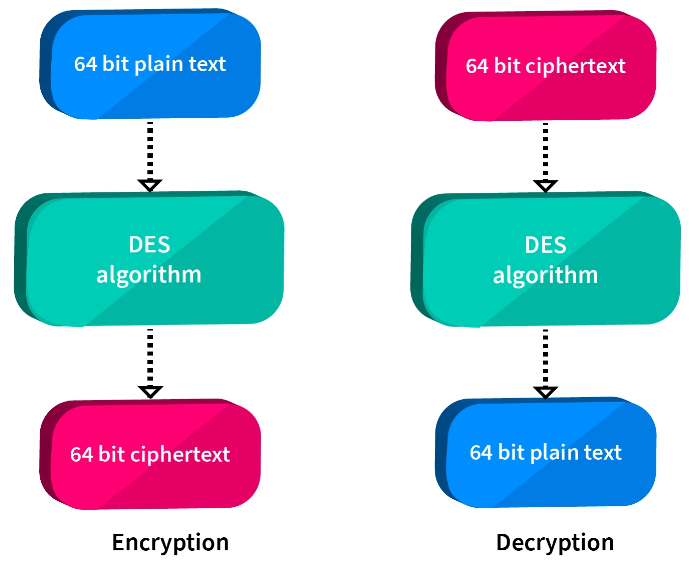
**Description:**

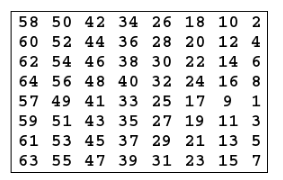
**Perform encryption and decryption using Data Encryption Standard**

The Data Encryption Standard algorithm is a block cipher algorithm that takes in 64-bit blocks of plaintext at a time as input and produces 64-bit blocks of cipher text at a time, using a 48-bit key for each input. In block cipher algorithms, the text to be encrypted is broken into ‘blocks’ of text, and each block is encrypted separately using the key.

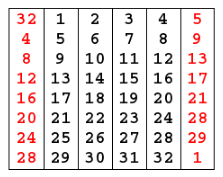


**Steps to perform Two rounds of DES**

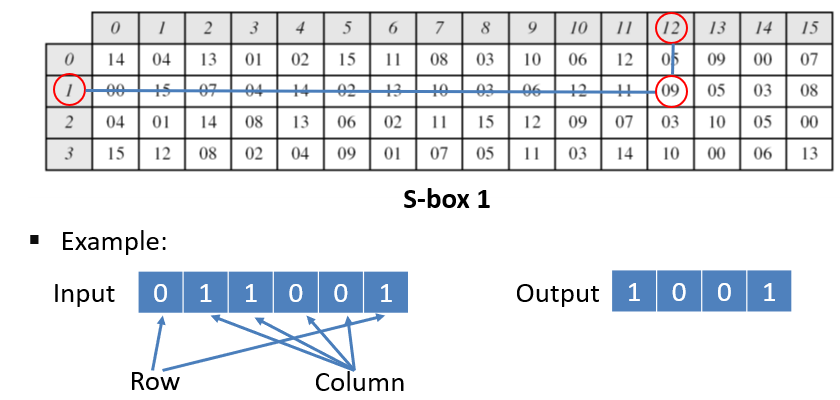
1. Take the 6 characters at a time from the file for encryption i.e. [8bitX6=64 bit block size]
2. Call a function that converts each character to ASCII and returns the binary of that character
3. Now we have 64 bit, call initial permutation function that changes the position of the bits.



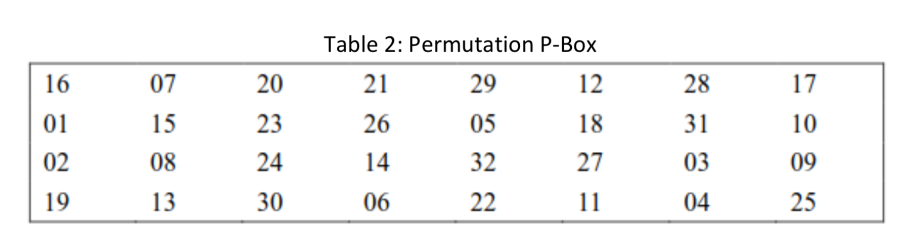
1. Take 56 bit key and create two round key K1 and K2 by left shifting both Left half and Right half[consider only first 48 bit from 56 bit for round key]
2. Expansion Permutation (E-table), Right half is expanded from 32-bits to 48-bits



1. S-box Substitution: Accepts 48-bits from XOR operation and produces 32-bits using 8 substitution boxes (each S-box has a 6-bit i/p and 4-bit o/p).

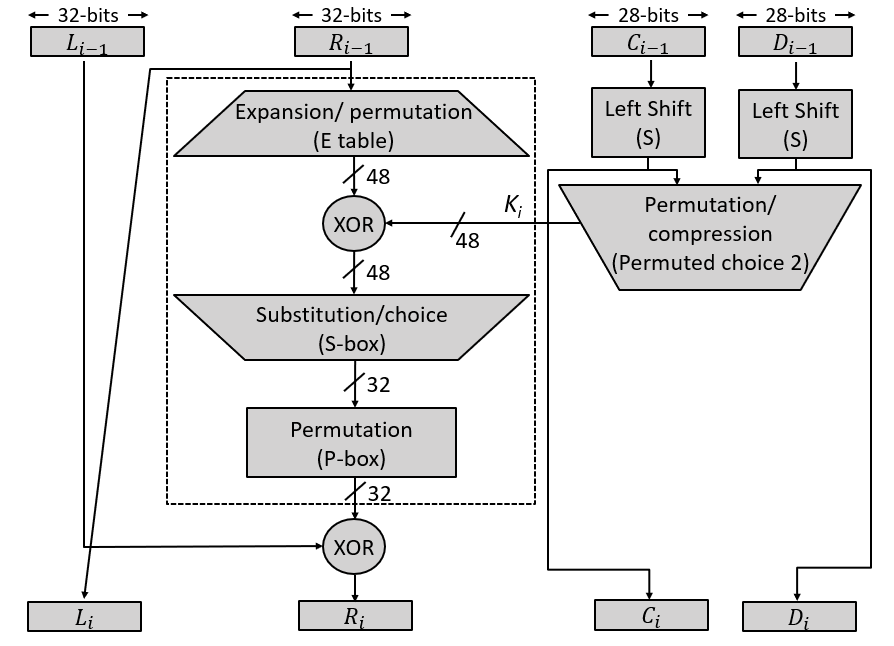


1. P-Box Permutation: rearrange the 32 bits in this fashion and read it again



1. XOR and Swap: Apply XOR Left portion and output of P-Box will be used as right portion for next round

**Single round DES**



**METHODOLOGY FOLLOWED:**

**File1 : decryption.cpp**

#include <iostream>

#include <bits/stdc++.h>

#include <string>

#include <fstream>

using namespace std;

string char\_to\_binaryString(int a)

{

    string st = "";

    while (a != 0)

    {

        st.push\_back(a % 2 + '0');

        a /= 2;

    }

    int n = 8 - st.size();

    for (int i = 0; i < n; i++)

    {

        st.append("0");

    }

    reverse(st.begin(), st.end());

    return st;

}

// IP -> initial prmutation array;

int IP[64] = {

    58, 50, 42, 34, 26, 18, 10, 2,

    60, 52, 44, 36, 28, 20, 12, 4,

    62, 54, 46, 38, 30, 22, 14, 6,

    64, 56, 48, 40, 32, 24, 16, 8,

    57, 49, 41, 33, 25, 17, 9, 1,

    59, 51, 43, 35, 27, 19, 11, 3,

    61, 53, 45, 37, 29, 21, 13, 5,

    63, 55, 47, 39, 31, 23, 15, 7};

string text64\_to\_initial\_permutation(string text64)

{

    string temp;

    for (int i = 0; i < 64; i++)

    {

        temp.push\_back(text64[IP[i] - 1]);

    }

    return temp;

}

int E[48] = {

    32, 1, 2, 3, 4, 5,

    4, 5, 6, 7, 8, 9,

    8, 9, 10, 11, 12, 13,

    12, 13, 14, 15, 16, 17,

    16, 17, 18, 19, 20, 21,

    20, 21, 22, 23, 24, 28,

    24, 25, 26, 27, 28, 29,

    28, 29, 30, 31, 32, 1};

string expantion\_32\_to\_48(string st)

{

    string temp;

    for (int i = 0; i < 48; i++)

    {

        temp.push\_back(st[E[i] - 1]);

    }

    return temp;

}

// binary string to int

int bsti(string st)

{

    int a = 0;

    int p = 1;

    for (int i = st.length() - 1; i >= 0; i--)

    {

        if (st[i] == '1')

        {

            a += p;

        }

        p \*= 2;

    }

    return a;

}

void cheack\_permutation(int a[])

{

    unordered\_map<int, int> m;

    for (int i = 0; i < 64; i++)

        m[a[i]]++;

    for (int i = 0; i < 64; i++)

    {

        if (m[i] > 1)

        {

            cout << "duplicate";

            break;

        }

    }

}

int pbox[32] = {16, 7, 20, 21, 29, 12, 28, 17, 1, 15, 23, 26, 5, 18, 31, 10, 2, 8, 24, 14, 32, 27, 3, 9, 19, 13, 30, 6, 22, 11, 4, 25};

string pboxPermutation(string st)

{

    string temp = "";

    for (int i = 0; i < 32; i++)

    {

        temp.push\_back(st[pbox[i] - 1]);

    }

    return temp;

}

string XOR(string st1, string st2)

{

    string temp;

    for (int i = 0; i < st1.size(); i++)

    {

        if (st1[i] == st2[i])

        {

            temp.push\_back('0');

        }

        else

        {

            temp.push\_back('1');

        }

    }

    return temp;

}

int sbox[4][16] = {

    {14, 4, 13, 1, 2, 15, 11, 8, 3, 10, 6, 12, 5, 9, 0, 7},

    {0, 15, 7, 4, 14, 2, 13, 10, 3, 6, 12, 11, 9, 5, 3, 8},

    {4, 1, 14, 8, 13, 6, 2, 11, 15, 12, 9, 7, 3, 10, 5, 0},

    {15, 12, 8, 2, 4, 9, 1, 7, 5, 11, 3, 14, 10, 0, 6, 13}};

string compress\_48\_to\_32(string st)

{

    string cmp = "";

    for (int i = 0; i < 8; i++)

    {

        string stx = st.substr(6 \* i, 6);

        int col = bsti(stx.substr(1, 4));

        int row = bsti(stx.substr(0, 1) + stx.substr(5, 1));

        // char\_to\_binaryString(sbox[row][col])->it gives 8 bit binary string like 9-> 00001001.

        // we wants only 4bit (0-15) hence we

        cmp.append(char\_to\_binaryString(sbox[row][col]).substr(4, 4));

    }

    return cmp;

}

string function1(string text32, string key)

{

    // expantion

    string Etext48 = expantion\_32\_to\_48(text32);

    // xor whith key

    string xorSt = XOR(Etext48, key);

    cout << "\n       " << Etext48 << "\n";

    cout << "       " << key << "\n";

    cout << "xorSt: " << xorSt << "\n";

    //  compretion / sunstitution /choise S-box

    string cmp = compress\_48\_to\_32(xorSt);

    return pboxPermutation(cmp);

}

int IIP[64] = {

    40, 8, 48, 16, 56, 24, 64, 32,

    39, 7, 47, 15, 55, 23, 63, 31,

    38, 6, 46, 14, 54, 22, 62, 30,

    37, 5, 45, 13, 53, 21, 61, 29,

    36, 4, 44, 12, 52, 20, 60, 28,

    35, 3, 43, 11, 51, 19, 59, 27,

    34, 2, 42, 10, 50, 18, 58, 26,

    33, 1, 41, 9, 49, 17, 57, 25};

string inverse\_initial\_permutation(string st)

{

    string temp;

    for (int i = 0; i < 64; i++)

    {

        temp.push\_back(st[IIP[i] - 1]);

    }

    return temp;

}

ifstream fin2;

string Decryption(string text64)

{

    // initial permutation

    string initial\_permutation = text64\_to\_initial\_permutation(text64);

    cout << "\nIP: " << initial\_permutation << " ";

    // left - right

    string L1 = initial\_permutation.substr(0, 32);

    string R1 = initial\_permutation.substr(32, 32);

    string key1;

    getline(fin2, key1);

    string key2;

    getline(fin2, key2);

    string L2 = R1;

    string fR1 = function1(R1, key2);

    string R2 = XOR(L1, fR1);

    // for round2

    string L3 = R2;

    string fR2 = function1(R2, key1);

    string R3 = XOR(L2, fR2);

    //  L3+R3 (-SWAP->) R3+L3  , HENCE WE RETURN R3+L3

    return R3 + L3;

}

int main()

{

    ifstream fin;

    fin.open("output.txt");

    fin2.open("key.txt");

    ofstream fout;

    fout.open("doutput.txt");

    string st;

    unordered\_map<string, char> m;

    unordered\_map<char, string> binary;

    for (int i = 0; i < 256; i++)

    {

        char ch = i;

        binary[ch] = char\_to\_binaryString(i);

        m[binary[ch]] = ch;

    }

    char ch;

    string plainText64 = "";

    int c = 0;

    while (fin.get(ch))

    {

        plainText64.append(binary[ch]);

        c++;

        if (c == 8)

        {

            string ciphertext = "";

            string str = Decryption(plainText64);

            string binary\_ciphetext = inverse\_initial\_permutation(str);

            for (int i = 0; i < 8; i++)

            {

                int a = i \* 8;

                char ch = m[binary\_ciphetext.substr(i \* 8, 8)];

                ciphertext.push\_back(ch);

            }

            fout << ciphertext;

            c = 0;

            plainText64 = "";

        }

    }

    fin.close();

    fout.close();

    fin2.close();

}

**File2: DES.cpp**

#include <iostream>

#include <bits/stdc++.h>

#include <string>

#include <fstream>

using namespace std;

string char\_to\_binaryString(int a)

{

    string st = "";

    while (a != 0)

    {

        st.push\_back(a % 2 + '0');

        a /= 2;

    }

    int n = 8 - st.size();

    for (int i = 0; i < n; i++)

    {

        st.append("0");

    }

    reverse(st.begin(), st.end());

    return st;

}

// IP -> initial prmutation array;

int IP[64] = {

    58, 50, 42, 34, 26, 18, 10, 2,

    60, 52, 44, 36, 28, 20, 12, 4,

    62, 54, 46, 38, 30, 22, 14, 6,

    64, 56, 48, 40, 32, 24, 16, 8,

    57, 49, 41, 33, 25, 17, 9, 1,

    59, 51, 43, 35, 27, 19, 11, 3,

    61, 53, 45, 37, 29, 21, 13, 5,

    63, 55, 47, 39, 31, 23, 15, 7};

string text64\_to\_initial\_permutation(string text64)

{

    string temp;

    for (int i = 0; i < 64; i++)

    {

        temp.push\_back(text64[IP[i] - 1]);

    }

    return temp;

}

int E[48] = {

    32, 1, 2, 3, 4, 5,

    4, 5, 6, 7, 8, 9,

    8, 9, 10, 11, 12, 13,

    12, 13, 14, 15, 16, 17,

    16, 17, 18, 19, 20, 21,

    20, 21, 22, 23, 24, 28,

    24, 25, 26, 27, 28, 29,

    28, 29, 30, 31, 32, 1};

string expantion\_32\_to\_48(string st)

{

    string temp;

    for (int i = 0; i < 48; i++)

    {

        temp.push\_back(st[E[i] - 1]);

    }

    return temp;

}

// binary string to int

int bsti(string st)

{

    int a = 0;

    int p = 1;

    for (int i = st.length() - 1; i >= 0; i--)

    {

        if (st[i] == '1')

        {

            a += p;

        }

        p \*= 2;

    }

    return a;

}

void cheack\_permutation(int a[])

{

    unordered\_map<int, int> m;

    for (int i = 0; i < 64; i++)

        m[a[i]]++;

    for (int i = 0; i < 64; i++)

    {

        if (m[i] > 1)

        {

            cout << "duplicate";

            break;

        }

    }

}

int pbox[32] = {16, 7, 20, 21, 29, 12, 28, 17, 1, 15, 23, 26, 5, 18, 31, 10, 2, 8, 24, 14, 32, 27, 3, 9, 19, 13, 30, 6, 22, 11, 4, 25};

string pboxPermutation(string st)

{

    string temp = "";

    for (int i = 0; i < 32; i++)

    {

        temp.push\_back(st[pbox[i] - 1]);

    }

    return temp;

}

string XOR(string st1, string st2)

{

    string temp;

    for (int i = 0; i < st1.size(); i++)

    {

        if (st1[i] == st2[i])

        {

            temp.push\_back('0');

        }

        else

        {

            temp.push\_back('1');

        }

    }

    return temp;

}

int sbox[4][16] = {

    {14, 4, 13, 1, 2, 15, 11, 8, 3, 10, 6, 12, 5, 9, 0, 7},

    {0, 15, 7, 4, 14, 2, 13, 10, 3, 6, 12, 11, 9, 5, 3, 8},

    {4, 1, 14, 8, 13, 6, 2, 11, 15, 12, 9, 7, 3, 10, 5, 0},

    {15, 12, 8, 2, 4, 9, 1, 7, 5, 11, 3, 14, 10, 0, 6, 13}};

string compress\_48\_to\_32(string st)

{

    string cmp = "";

    for (int i = 0; i < 8; i++)

    {

        string stx = st.substr(6 \* i, 6);

        int col = bsti(stx.substr(1, 4));

        int row = bsti(stx.substr(0, 1) + stx.substr(5, 1));

        // char\_to\_binaryString(sbox[row][col])->it gives 8 bit binary string like 9-> 00001001.

        // we wants only 4bit (0-15) hence we

        cmp.append(char\_to\_binaryString(sbox[row][col]).substr(4, 4));

    }

    return cmp;

}

string function1(string text32, string key)

{

    // expantion

    string Etext48 = expantion\_32\_to\_48(text32);

    // xor whith key

    string xorSt = XOR(Etext48, key);

    //  compretion / sunstitution /choise S-box

    string cmp = compress\_48\_to\_32(xorSt);

    return pboxPermutation(cmp);

}

int IIP[64] = {

    40, 8, 48, 16, 56, 24, 64, 32,

    39, 7, 47, 15, 55, 23, 63, 31,

    38, 6, 46, 14, 54, 22, 62, 30,

    37, 5, 45, 13, 53, 21, 61, 29,

    36, 4, 44, 12, 52, 20, 60, 28,

    35, 3, 43, 11, 51, 19, 59, 27,

    34, 2, 42, 10, 50, 18, 58, 26,

    33, 1, 41, 9, 49, 17, 57, 25};

string inverse\_initial\_permutation(string st)

{

    string temp;

    for (int i = 0; i < 64; i++)

    {

        temp.push\_back(st[IIP[i] - 1]);

    }

    return temp;

}

ofstream key\_file\_out;

string Encryption(string text64)

{

    // initial permutation

    string initial\_permutation = text64\_to\_initial\_permutation(text64);

    // left - right

    string L1 = initial\_permutation.substr(0, 32);

    string R1 = initial\_permutation.substr(32, 32);

    // key\_permutation\_64;

    string key\_56 = "";

    for (int i = 0; i < 56; i++)

    {

        char ch = rand() % 2 + '0';

        key\_56.push\_back(ch);

    }

    //  cout << "\n56 key permutation: " << key\_56 << "\n";

    // initial 56 bit permutation into right  & left part

    string lkey\_0 = key\_56.substr(0, 28);  // left 28 bit of 56 bit permutation1

    string rkey\_0 = key\_56.substr(28, 28); // rigt 28 bit of 56 bit permutation1

    // for round1

    string lkey\_1 = lkey\_0.substr(1, 28) + lkey\_0.substr(0, 1); // circular left shift on lkey

    string rkey\_1 = rkey\_0.substr(1, 28) + rkey\_0.substr(0, 1); // circular left shift on rkey

    string key1 = (lkey\_1 + rkey\_1).substr(0, 48);              // first 48 bits form 56 bit    lkey\_1 + rkey\_2

    string L2 = R1;

    string fR1 = function1(R1, key1);

    string R2 = XOR(L1, fR1);

    key\_file\_out << key1 << "\n";

    // cout << "L2 : " << L2 << " R2: " << R2 << "\n";

    // for round2

    string lkey\_2 = lkey\_1.substr(1, 28) + lkey\_1.substr(0, 1); // circular left shift on lkey

    string rkey\_2 = rkey\_1.substr(1, 28) + rkey\_1.substr(0, 1); // circular left shift on rkey

    string key2 = (lkey\_2 + rkey\_2).substr(0, 48); // first 48 bits form 56 bit    lkey\_2 + rkey\_2

    key\_file\_out << key2 << "\n";

    string L3 = R2;

    string fR2 = function1(R2, key2);

    string R3 = XOR(L2, fR2);

    //  cout << "L3 : " << L3 << " R3: " << R3 << "\n";

    //  L3+R3 (-SWAP->) R3+L3  , HENCE WE RETURN R3+L3

    return R3 + L3;

}

int main()

{

    ifstream fin;

    fin.open("input.txt");

    ofstream fout;

    fout.open("output.txt");

    key\_file\_out.open("key.txt");

    string st;

    unordered\_map<string, char> m;

    unordered\_map<char, string> binary;

    for (int i = 0; i < 256; i++)

    {

        char ch = i;

        string s = char\_to\_binaryString(i);

        binary[ch] = s;

        m[s] = ch;

    }

    char ch;

    int c = 0;

    string plainText64 = "";

    int cnt = 0;

    while (fin.get(ch))

    {

        cnt++;

        plainText64.append(binary[ch]);

        c++;

        if (c == 8)

        {

            string ciphertext = "";

            string str = Encryption(plainText64);

            string binary\_ciphetext = inverse\_initial\_permutation(str);

            for (int i = 0; i < 8; i++)

            {

                char ch = m[binary\_ciphetext.substr(i \* 8, 8)];

                ciphertext.push\_back(ch);

            }

            fout << ciphertext;

            c = 0;

            plainText64 = "";

        }

    }

    cout << "\n\*\*\*\*\*\*\*\*\*\*\*\*" << cnt << "\n";

    if (c != 0)

    {

        string chiphertext = "";

        int sz = plainText64.length();

        for (int i = sz / 8; i < 8; i++)

        {

            plainText64.append(binary[' ']);

        }

        string str = Encryption(plainText64);

        string binary\_ciphetext = inverse\_initial\_permutation(str);

        for (int i = 0; i < 8; i++)

        {

            char ch = m[binary\_ciphetext.substr(i \* 8, 8)];

            chiphertext.push\_back(ch);

        }

        fout << chiphertext;

    }

    fin.close();

    fout.close();

    key\_file\_out.close();

}

**File3: input.txt**

**hello how jjbkjbjkbbbbbbjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjj**

**kkkkkkkkkkkkkkkkkkkkk nirma kkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkk**

**llllllllllllllllllllllllllllllllllllllllllllllllllllllllllllllllllllllllllllllllllllllllll**

**hello how jjbkjbjkbbbbbbjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjj**

**kkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkk**

**llllllllllllllllllllllllllllllllllllllllllllllllllllllllllllllllllllllllllllllllllllllllll**

**hello how jjbkjbjkbbbbbbjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjj**

**kkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkk**

**llllllllllllllllllllllllllllllllllllllllllllllllllllllllllllllllllllllllllllllllllllllllll**

**File4: key.txt (these are keys through which we encrypted the message. these keys are generated during performing encryption. we store these keys in key.txt file for decrypt the message.)**

**100100000111111101010010010111010101110110110111**

**001000001111111010100100101110101011101101101110**

**110010000000001010001101100100010010110001111100**

**100100000000010100011011001100100101100011111000**

**001111000101110100010001111011111101000001001010**

**011110001011101000100011110011111010000010010101**

**010000101001011000011010111011010110110010001101**

**100001010010110000110101110010101101100100011011**

**000000110110000010101100100010000111000100111100**

**000001101100000101011001000000001110001001111001**

**011110101001100101101001101101001111011110111100**

**111101010011001011010011011010011110111101111001**

**101111100011010001000111010010110001101000011010**

**011111000110100010001110100101100011010000110101**

**110110110100100110111101011001111000000101000111**

**101101101001001101111010110111110000001010001110**

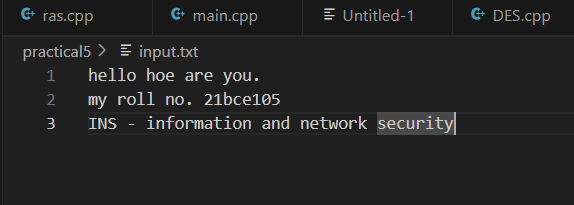
**011000010011000100100010101100100111011101000101**

**110000100110001001000101011001001110111010001011**

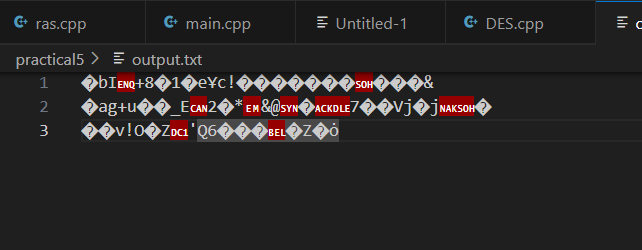
**011111000010101010011100110001011100010101010001**

**111110000101010100111001100010111000101010100011**

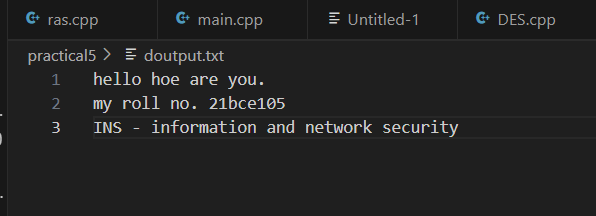
**File5: input.txt**

****

**File6: output.txt**

****

**File7: doutput.txt**

****